Data Loading, Storage, and File Formats

CS3753 Data Science

Prof. Weining Zhang

Topics

The Pandas package provides functions to load DataFrame from and to save DataFrame to various type of files including

- Text files
- JSON document
- XML and HTML web pages
- Binary files
- SQL databases

In [ ]:
```
from __future__ import division
from numpy.random import randn
import numpy as np
import os
import sys
import matplotlib.pyplot as plt
np.random.seed(12345)
plt.rc('figure', figsize=(10, 6))
from pandas import Series, DataFrame
import pandas as pd
np.set_printoptions(precision=4)
```

Reading and Writing Data in Text Format

```
pd.read_csv(file, sep=',', <parameters>)
pd.read_table(file, <parameters>)
pd.to_csv(file, <parameters>)
```

- Behavior is controlled by many parameters

In [ ]:
```
!cat ch06/ex1.csv
```

In [ ]:
```
df = pd.read_csv('ch06/ex1.csv')
df
```

In [ ]:
```
pd.read_table('ch06/ex1.csv', sep=',')
```

Add or Change Column Names

```
pd.read_csv(file, header=None, names=listOfNames,...)
```

- Header specifies how to identify or infer header
- Names assignes column names

In [ ]:
```
!cat ch06/ex2.csv
```

In [ ]:
```
pd.read_csv('ch06/ex2.csv', header=None)
pd.read_csv('ch06/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

In [ ]:
```
df1.iloc[1]
```

Designate Index Columns

```
pd.read_csv(file, index_col=listOfColumns, ...)
```

- Index_col identifies columns used to label rows

In [ ]:
```
names = ['a', 'b', 'c', 'd', 'message']
pd.read_csv('ch06/ex2.csv', names=names, index_col='message')
```

In [ ]:
```
!cat ch06/csv_mindex.csv
parsed = pd.read_csv('ch06/csv_mindex.csv', index_col=['key1', 'key2
'])
parsed
```

In [ ]:
```
parsed.loc['one','c']['value1']
```

Specify Alternative Separator

- Use parameter sep or delimiter

In [ ]:
```
list(open('ch06/ex3.txt'))
```

In [ ]:
```
result = pd.read_table('ch06/ex3.txt', sep='\s+')
result
```

In [ ]:
```
!cat ch06/ex4.csv
pd.read_csv('ch06/ex4.csv', skiprows=[0, 2, 3])
```

Filling or Replacing Null Values

```
pd.read_csv(file, na_values=...)
```

- Additional strings to recognize as NA/NaN.
- If dict passed, specific per-column NA values.

In [ ]:
```
!cat ch06/ex5.csv
result = pd.read_csv('ch06/ex5.csv')
result
```

In [ ]:
```
pd.isnull(result)
```

In [ ]:
```
result = pd.read_csv('ch06/ex5.csv', na_values='Unknown')
result
```

In [ ]:
```
sentinels = {'message': ['foo', 'NA'], 'something': ['two']}
pd.read_csv('ch06/ex5.csv', na_values=sentinels)
```

Read Text Files in Pieces

nrows=
chunksize=

In [ ]:
```
result = pd.read_csv('ch06/ex6.csv')
result[:10]
```

In [ ]:
```
pd.read_csv('ch06/ex6.csv', nrows=5)
```

In [ ]:
```
chunker = pd.read_csv('ch06/ex6.csv', chunksize=1000)
chunker
```

In [ ]:
```
chunker = pd.read_csv('ch06/ex6.csv', chunksize=1000)

tot = Series([])
for piece in chunker:
    print(piece)
    tot = tot.add(piece['key'].value_counts(), fill_value=0)

tot = tot.sort_values(ascending=False)
```

In [ ]:
```
tot[:10]
```

Write Data Out to Text Format

In [ ]:
```
data = pd.read_csv('ch06/ex5.csv')
data
```

In [ ]:
```
data.to_csv('ch06/out.csv')
!cat ch06/out.csv
```

In [ ]:
```
data.to_csv(sys.stdout, sep='|')
```

In [ ]:
```
data.to_csv(sys.stdout, na_rep='NULL')
```

In [ ]:
```
data.to_csv(sys.stdout, index=False, header=False)
```

In [ ]:
```
data.to_csv(sys.stdout, index=False, columns=['a', 'b', 'c'])
```

In [ ]:
```
dates = pd.date_range('1/1/2000', periods=7)
print(dates)
ts = Series(np.arange(7), index=dates)
ts.to_csv('ch06/tseries.csv')
!cat ch06/tseries.csv
```

In [ ]:
```
Series.from_csv('ch06/tseries.csv', parse_dates=True)
```

Manually Working with Delimited Formats

The following example use csv package to read a csv file into list of lines, stripping "" of values. It then rebuild the table with a customerized set of dialect

In [ ]:
```
!cat ch06/ex7.csv
```

In [ ]:
```
import csv
f = open('ch06/ex7.csv')

reader = csv.reader(f)
```

In [ ]:
```
for line in reader:
    print(line)
```

In [ ]:
```
lines = list(csv.reader(open('ch06/ex7.csv')))
print(lines)
header, values = lines[0], lines[1:]
data_dict = {h: v for h, v in zip(header, zip(*values))}
data_dict
```

In [ ]:
```
class my_dialect(csv.Dialect):
    lineterminator = '\n'
    delimiter = ';'
    quotechar = '"'
    quoting = csv.QUOTE_MINIMAL
```

In [ ]:
```
with open('mydata.csv', 'w') as f:
    writer = csv.writer(f, dialect=my_dialect)
    writer.writerow(('one', 'two', 'three'))
    writer.writerow(('1', '2', '3'))
    writer.writerow(('4', '5', '6'))
    writer.writerow(('7', '8', '9'))
```

In [ ]:
```
%cat mydata.csv
```

JSON (JavaScript Object Notation) Data

- Data is formatted as a JSON string, can be transmitted over HTTP protocol
- Use json package

  `json.loads: convert a JSON string to a Python object`
  `json.dumps: convert a Python object to a JSON string`

- Use pandas functions

  `pd.read_json: load JSON data into DataFrame`
  `pd.to_json: write DataFrame into JSON file`

In [ ]:
```
obj = """
{"name": "Wes",
 "places_lived": ["United States", "Spain", "Germany"],
 "pet": null,
 "siblings": [{"name": "Scott", "age": 25, "pet": ["Zeus", "Zuko"]},
              {"name": "Katie", "age": 33,
               "pet": ["Sixes", "Stache", "Cisco"]}]
}
"""
```

In [ ]:
```
import json
result = json.loads(obj)
result
```

In [ ]:
```
asjson = json.dumps(result)
asjson
```

In [ ]:
```
siblings = DataFrame(result['siblings'], columns=['name', 'age', 'pe
t'])
siblings
```

In [ ]:
```
with open('ch06/product.json') as f:
   data = json.load(f)
print(data)
```

In [ ]:
```
%cat ch06/product.json
```

In [ ]:
```
data = pd.read_json('ch06/product.json')
data
```

Read Data from HTML, XML, and Web Sources

- Experiment with Beautiful Soup 4

```
import bs4.BeautifulSoup, urllib.request,
       lxml.objectify
```

  - Open the URL of an HTML document (a web page)
  - Parse it into an object representing the structure of HTML doc
  - Access elements and attributes by their paths

In [ ]:
```
from bs4 import BeautifulSoup
import urllib.request
```

Access Elements and Attributes

```
doc.<element>.<element>
doc.<element>....<element>.<attribute>
doc.find_all(<element>|<attribute>)
```

- Based on the concept of XPath patterns
- Can also find title, parent, etc.

Example: Extract a Data Table from Yahoo Finance

- Get the calls from the Apple (AAPL) realtime price quote
- The web page contains two tables: Calls and Puts
- Many rows, each contains 11 fields

In [ ]:
```
with urllib.request.urlopen('http://finance.yahoo.com/q/op?s=AAPL+Options') as response:
   html = response.read()
soup = BeautifulSoup(html)
print(soup.prettify())
```

In [ ]:
```
soup.title
```

In [ ]:
```
soup.title.name
```

In [ ]:
```
soup.title.string
```

In [ ]:
```
soup.title.parent.name
```

In [ ]:
```
soup.find_all('p')
```

In [ ]:
```
soup.a
```

In [ ]:
```
soup.find_all('a')
```

In [ ]:
```
links = soup.find_all('a')
```

In [ ]:
```
links[15:20]
```

In [ ]:
```
lnk = links[28]
```

In [ ]:
```
lnk
```

In [ ]:
```
type(lnk)
```

In [ ]:
```
lnk.name
```

In [ ]:
```
lnk['href']
```

In [ ]:
```
lnk.contents
```

In [ ]:
```
urls = [lnk['href'] for lnk in links]
urls[-10:]
```

Steps

- Get a list of tables
- Find the Calls table
- Get the header labels of the Calls table
- Create a calldf DataFrame with appropppriate column names and enough rows
- Get a list of rows in Calls table
- Extract data fields of a row, add a new row to calldf

In [ ]:
```
tables = soup.find_all('table')
tables # two tables: calls and puts
```

In [ ]:
```
len(tables)
```

In [ ]:
```
calls = tables[0]
ths = calls.find_all('th')
ths
```

In [ ]:
```
header = []
for h in ths :
    header.extend(h.span.contents)
header
```

In [ ]:
```
trs = calls.find_all('tr')
calldf = pd.DataFrame(index=range(len(trs)-1), columns=header)
print(calldf)
```

Extract Rows and Add to DataFrame

In [ ]:
```
# Extract rows and insert into calldf DataFrame
i = 0
for t in trs[1:] :
    l = []
    ds = t.find_all('td')
    # print('ds=', ds)
    l.extend(ds[0].a.contents)
    l.extend(ds[1].contents)
    l.extend(ds[2].a.contents)
    l.extend(ds[3].contents)
    l.extend(ds[4].contents)
    l.extend(ds[5].contents)
    l.extend(ds[6].span.contents)
    l.extend(ds[7].span.contents)
    l.extend(ds[8].contents)
    l.extend(ds[9].contents)
    l.extend(ds[10].contents)
    #print(l)
    calldf.iloc[i] = pd.Series(l, index=header)
    i = i+1

calldf
```

Parsing XML with lxml.objectify

```
import lxml.objectify
```

In [ ]:
```
%cd ch06
```

In [ ]:
```
!head -21 Performance_MNR.xml
```

In [ ]:
```
from lxml import objectify

path = 'Performance_MNR.xml'
parsed = objectify.parse(open(path))
root = parsed.getroot()
root
```

In [ ]:
```
data = []

skip_fields = ['PARENT_SEQ', 'INDICATOR_SEQ',
               'DESIRED_CHANGE', 'DECIMAL_PLACES']

for elt in root.INDICATOR:
    el_data = {}
    for child in elt.getchildren():
        if child.tag in skip_fields:
            continue
        el_data[child.tag] = child.pyval
    data.append(el_data)
data
```

In [ ]:
```
perf = DataFrame(data)
perf
```

In [ ]:
```
root
```

In [ ]:
```
root.get('href')
```

In [ ]:
```
root.text
```

Binary Data Formats

- A Python object, such as DataFrame, can be serialized and stored in a binary file

  `pd.to_pickle(file), pd.from_pickle(file)`

  - compression, such as gzip and zip, can be specified
- Another binary format is HDF5, designed to store large numerical arrays of homogenous type.
  - Use pandas HDFStore object to create binary file, store and retreive data

Using Pickle (Serialization)

In [ ]:
```
#%cd /Users/wzhang/teaching/cs3753/dataSciCourse/lecture-code/week05
```

In [ ]:
```
frame = pd.read_csv('ch06/ex1.csv')
frame
```

In [ ]:
```
frame.to_pickle('ch06/frame_pickle')
!cat 'ch06/frame_pickle'
```

In [ ]:
```
pd.read_pickle('ch06/frame_pickle')
```

Using HDFStore

- An HDFStore object stores data as dictionary

```
hfd = pd.HDFStore('file.h5')
hfd.put(...)
hfd.append(...)
hfd[newKey] = dataTable
hfd.close(...)
```

In [ ]:
```
store = pd.HDFStore('mydata.h5')
store['data1'] = frame
store['data2'] = frame[['a', 'c']] * 3
store.keys()
```

In [ ]:
```
store['data1']
```

In [ ]:
```
store['data2']
```

In [ ]:
```
store.close()
!cat mydata.h5
```

In [ ]:
```
os.remove('mydata.h5')
```

Interacting with HTML and Web APIs

In [ ]:
```
import requests
url = 'https://api.github.com/repos/pydata/pandas/milestones/28/labe
ls'
resp = requests.get(url)
resp
```

In [ ]:
```
data = resp.json()
```

In [ ]:
```
data[:5]
```

In [ ]:
```
issue_labels = DataFrame(data)
issue_labels
```

Interacting with SQL Databases

- Import package that is required for specific DBMS
- Connect to DBMS
- Ask DBMS to execute a query
- Get result from DBMS server and store in DataFrame
- Further processing data in Python

Example

- Connect to an in-mamery SQLite database
- Create a data table
- Insert some tuples into the table
- Run a database SQL query to retreive data

In [ ]:
```
import sqlite3

query = """
CREATE TABLE test
(a VARCHAR(20), b VARCHAR(20),
 c REAL,        d INTEGER
);"""

con = sqlite3.connect(':memory:')
con.execute(query)
con.commit()
```

In [ ]:
```
data = [('Atlanta', 'Georgia', 1.25, 6),
        ('Tallahassee', 'Florida', 2.6, 3),
        ('Sacramento', 'California', 1.7, 5)]
stmt = "INSERT INTO test VALUES(?, ?, ?, ?)"

con.executemany(stmt, data)
con.commit()
```

In [ ]:
```
cursor = con.execute('select * from test')
rows = cursor.fetchall()
rows
```

In [ ]:
```
cursor.description
```

In [ ]:
```
DataFrame(rows, columns=tuple(zip(*cursor.description))[0])
```

In [ ]:
```
import pandas.io.sql as sql
sql.read_sql('select * from test', con)
```